UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/700,338	11/03/2003	Lewis K. Cirne	WILY-01013US0	5180

28554 7590 09/13/2011

Vierra Magen Marcus & DeNiro LLP
575 Market Street, Suite 2500
San Francisco, CA 94105

EXAMINER	
WEI, ZHENG	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
09/13/2011	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/700,338	CIRNE ET AL.	
	Examiner	Art Unit	
	ZHENG WEI	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *17 December 2009*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ An election was made by the applicant in response to a restriction requirement set forth during the interview on _____; the restriction requirement and election have been incorporated into this action.

4)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

5)☒ Claim(s) *1-3,5-13,17-23,28-30,32-35,37-42,44-47,51 and 52* is/are pending in the application.

5a) Of the above claim(s) _____ is/are withdrawn from consideration.

6)☐ Claim(s) _____ is/are allowed.

7)☒ Claim(s) *1-3,5-13,17-23,28-30,32-35,37-42,44-47,51 and 52* is/are rejected.

8)☐ Claim(s) _____ is/are objected to.

9)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

10)☐ The specification is objected to by the Examiner.

11)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

12)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

13)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

Detailed Action

Remarks

1. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on 12/17/2009 has been entered.
2. This office action is in response to the amendment filed on 12/17/2009.
3. Claims 1-3, 5-6, 13, 17-18, 22-23, 33-35, 37, 39-42, 47 and 52 have been amended.
4. Claims 14-15, 24, 26-27, 49-50 and 53-59 have been cancelled.
5. 35 U.S.C. § 112 rejection to claims 13-32 and 59 is withdrawn in view Applicants' amendment.
6. Claims 1-3, 5-13, 17-23, 28-30, 32-35, 37-42, 44-47 and 51-52 remain pending and have been examined.

Response to Arguments

7. Applicant's arguments filed on 12/17/2009, in particular on pages 12-17, have been fully considered but they are not persuasive. For example:
   - At page 15, last paragraph - page 16, first paragraph, regarding to claim 1, Applicants assert that "Berkley in view of Webster in further view of Grove

fails to discloses, 'modifying said method for a particular purpose only if said method calls another method,' as claimed". However, Examiner's position is that Webster discloses the facts that "[i]a general sense the running of a Java program represents a succession of Java method calls. The ability to track or monitor these method calls is very important to a Java programmer for diagnostic and debugging purpose" [emphasis added] (see for example, col.1, lines 39-42) and "[i]n order to provide trace information from a program about specific method calls, a user defines a selection of methods to be traced" [emphasis added] (see for example, ABSTRACT). Therefore, it can be seen that user can select the specific method to be traced including the method calls another method as addressed by Webster above and further modify such specific method by putting flag in method block (see for example, Fig.3, step 330, "CL Puts Flag in Method Block" and related text). One would have been motivated to selective modify and trace the specific method calls ("modifying said method for particular purpose only if said method calls another method") in order to "avoid the system having to output large quantities of irrelevant information, thereby greatly assisting in performance" [emphasis added] as also taught by Webster (see for example, col.2, lines 37-39).

- At page 14 – page 15, regarding to claim 13, Applicants assert that "Berkley in view of Webster in further view of Grove fails to discloses, 'using a first tracing mechanism for said methods that call one or more other methods and

are not synthetic without using said first tracing mechanism for methods that do not call one or more other methods or are synthetic,' as claimed". However, as discussed above, Webster discloses a method to selective trace (using or without using a "first tracing mechanism") method calls according to the users defined selection in their interests (see for example, col.2, lines 34-36, "A user can specify particular trace information of interest, and interpreter can then track for each called method whether or not trace information is to be provided"). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to define tracing or not tracing particular method calls according the user/programmer's interests including the methods call other method and/or the methods are synthetic methods as taught by Webster (see for example, col.2, lines 34-39).

- At pages 16-17, regarding to claim 22, Applicants assert that "Berkley in view of Webster in further view of Berry fails to disclose, 'modifying for a particular purpose only those methods that call one or more other methods and have an access level of either public or package in the JAVA programming language.' as claimed". However, as discussed above, Webster discloses a method to selective trace method calls according to the users defined selection in their interests (see for example, col.2, lines 34-36, "A user can specify particular trace information of interest, and interpreter can then track for each called method whether or not trace information is to be provided") and further modify such specific method by putting flag in method block (see for example, Fig.3,

step 330, "CL Puts Flag in Method Block" and related text). Moreover, Webster also discloses the tracing method is for diagnosing and debugging Java program in JVM (Java virtual machine) (see for example, col.1, lines 38-41 and col.3, lines 48-65 and related text). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to define the selection for tracing particular method calls including the methods call other method and/or have different access levels in Java program. One would have been motivated to do so to selective trace method calls according the user/programmer's interests as taught by Webster (see for example, col.2, lines 34-39).

- At page 15, regarding to claim 33, Applicants submit that "the prior art fails to teach or in any way suggest, 'means for tracing said method for a particular purpose only if said method calls another method, said method can be called by a sufficient scope of one or more other methods and said method is not a synthetic method.'". However, it is noted that applicants' cited "Currently amended Claim 33" as listed in page 15 is different from the actual amended claim 33 as filed on 12/17/2009 and thus Applicants' arguments above is not considered.
- At page 16, regarding to claim 33 (correct version of the amended claim 33) with the same limitation as recited in claim 1, please see the discussion above.

- At page 11-14, regarding to claim 47, Applicants assert that "the prior art fails to disclose, 'adding a tracer to said method only if said method is automatically determined to be complex,' as claimed. As recited in claim 47, a method is complex if the method satisfies the criteria that: 'said method calls another method,' 'said method has an access level of public or package in the JAVA programming language,' and 'said method is not flagged by a compiler as being synthetic.'" and there is no suggestion or motivation for tracing the specific method including "the combination of only methods that call other methods, methods that are not synthetic, and methods that access level of public or package in the JAVA programming language.", However, Examiner's position as addressed above is that Webster discloses a method to selective trace method calls according to the users defined selection in their interests (see for example, col.2, lines 34-36, "A user can specify particular trace information of interest, and interpreter can then track for each called method whether or not trace information is to be provided") and further modify such specific method by putting flag in method block (see for example, Fig.3, step 330, "CL Puts Flag in Method Block" and related text). It can be seen that the users can define/specify particular method that needs to be traced according to their interests "for the diagnostic and debugging purposes" in Java virtual machine environment. Furthermore, it is well-known in Java programming fields that the terms and concepts including Java method calls, Java method access levels (public or package) and Java

synthetic method are widely used in Java programming practice. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to selectively trace the specific method calls for the particular diagnostic/debugging purposes and further considering system's performance by avoiding system generating irrelevant information as suggested by Webster (see for example, col.1, lines 40-42, "The ability to track or monitor these method calls is very important to a Java programmer for diagnostic and debugging purpose" and col.2, lines 37-39, "avoid the system having to output large quantities of irrelevant information, thereby greatly assisting in performance").

- At page 12, regarding to claim 47, Applicants further pointed out "if the tracer were to be added to too many methods, then the performance of the software that contains the methods could be negatively impacted. However, if the tracer is not added to certain methods, then methods that should be traced might not be traced. It can be very challenging to automatically determine which method should be traced and which need not be traced. Applicants respectfully assert that the prior art provides no guidance that would suggest to one of ordinary skill in the art to add tracers to only the set of methods defined by the criteria recited in claim 47". However, as discussed above, Webster discloses the tracking/monitoring method calls in running Java program is very important and selectively trace specific/particular method as required can avoid the system having to output large quantities of irrelevant

information, thereby greatly assisting in performance. Therefore, Webster's diagnostic, debugging and performance purposes above provide motivation and suggestion for adding tracers to only the set of methods as recited in claim 47.

Claim Rejections - 35 USC § 103

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

9. Claims 1-3, 5-8, 10, 12-13, 17, 18, 20, 39-42, 44, 47 and 51-52 are rejected under 35 U.S.C. 103(a) as being unpatentable over Berkley (Berkley et al., US 6351843 B1) in view of Webster (US6,738,965 now cited as prior art in this office action) in further view of Grove (Grove et al., Call Graph Construction in Object-Oriented Languages, now is recorded as prior reference)

Claim 1:

Berkley discloses a process for monitoring, comprising:

- accessing a method (see for example Fig.5, step 350 and related text, also see, col.2, lines 36-37, "Further, the system includes means for running the application executable using the modified runtime configuration settings");
- automatically determining whether to modify said method, said step of automatically determining whether to modify said method (see for example,

Fig.1 and related text, "METHOD B", "METHOD C" and also see col.2, lines 39-40, "means for determining whether the function (method) is active for a class of the executable using the modified configuration settings"); and

- modifying said method for a particular purpose if said method calls another method. (see for example, col.2, lines 41-43, "means for dynamically creating a redirection stub to insert the function for the class if the function is active for that class")

- Berkley also discloses a method to provide configuration settings to specify user interested information/methods that need to be traced and further determining whether to modify said method to insert trace function according the user's settings (see for example, Fig.5, item 300 "Configuration Settings" item 310 "Add Setting to Specify trace for Desired Class:, item 320 "New Configuration Settings and related text). But Berkley does not explicitly disclose the determination includes automatically determining whether said method calls another method. However, Webster in the same analogous art of selective tracing methods discloses the similar solution that a user can specify particular trace information of interest and provides a selection of methods to be traced from a program (see for example, Summary and Fig.3 item 325 and related text). Grove discloses a method to construct a call graph (see for example, Fig.1 and related text; also see p.109, section 2.1, first paragraph). As Webster disclosed that different users have different purpose/interest to trace/debug different methods (see for example,

ABSRACT, "selection of method to be traced"), it is obvious said method also including those methods calling other methods as indicated by Grove's call graph. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to trace/debug user interested methods by identify/determining the method calling other method using Webster and Grove's method and further configuring and inserting tracing function using Berkley's method. One would have been motivated to do so to selectively trace the specific method calls for the particular diagnostic, debugging purposes and further considering system's performance by avoiding system generating irrelevant information as suggested by Webster (see for example, col.1, lines 40-42, "The ability to track or monitor these method calls is very important to a Java programmer for diagnostic and debugging purpose" and col.2, lines 37-39, "avoid the system having to output large quantities of irrelevant information, thereby greatly assisting in performance").

Claim 2:

Berkley discloses processes according to claims 1 and 13 above respectively, but does not explicitly disclose said step of determining whether to modify said method includes automatically determining whether said method is non-synthetic. However, It is well known in the Java programming that all synthetic methods generated by Java compiler are flagged in the class file and thus are easily

identified (A well-known and widely used Java programming standard: Java Virtual Machine Specification). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to automatically determine whether said method is non-synthetic by checking the synthetic attribute field in bytecode while being compiled by JIT, Hotspot runtime or other bytecode scanning tools. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 3:

Berkley discloses processes according to claim 1 and 13 above respectively, but does not explicitly disclose said step of automatically determining whether to modify aid method includes determining whether said method has an access level of public or package. However, it is well known in the Java programming that JVM specification (see for example, a well-known and widely used Java Virtual Machine Specification) defines a set of access flags in method_info structure which has a flag name "ACC_PUBLIC" for access level of public or package. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method has

an access level of public or package by using JIT, Hotspot runtime or other bytecode scanning tools to check this flag to determining whether said method has an access level of public or package. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claims 5 and 17:

Berkley discloses processes according to claim 1 and 13 above respectively, but does not disclose said step of automatically determining whether to modify said method includes determining whether said method is non-synthetic, calls another method and has an access level of public or package. However, it is well-known in Java programming art that the terms and concepts including Java method calls, Java method access levels (public or package) and Java synthetic method are widely used in Java programming practice. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to selectively trace the specific method calls for the particular diagnostic/debugging purposes and further considering system's performance by avoiding system generating irrelevant information as suggested by Webster (see for example, col.1, lines 40-42, "The ability to track or monitor these method calls

is very important to a Java programmer for diagnostic and debugging purpose" and col.2, lines 37-39, "avoid the system having to output large quantities of irrelevant information, thereby greatly assisting in performance").

Claims 6 and 18:

Berkley discloses processes according to claim 1 and 13 above respectively, but does not disclose said step of automatically determining whether to modify said method includes determining whether said method calls one or more different methods and can be called by a sufficient scope of one or more other methods. However, it is well known in the Java programming that JVM specification (see for example, a well-known and widely used Java Virtual Machine Specification) defines method by using a block starting with the tag "Method" that contains the information about calling other methods in java bytecode. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method calls another method and can be called by a sufficient scope of one or more other methods by checking the method information in that block while running by JIT in JVM or other bytecode scanning tools. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or

more methods for which the function is to be implemented")

Claim 7:

Berkley further discloses a process according to claim 1, wherein: said step of modifying includes modifying object code (see for example, col.2, lines 45-46, "inserting a function into an application executable without recompiling the executable.").

Claim 8:

Berkley also discloses a process according to claim 1, wherein: said step of modifying includes adding a tracer for said method (see for example, Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…").

Claim 10:

Berkley further discloses a process according to claim 1, wherein: said step of modifying includes adding exit code and start code to existing object code (see for example, Fig.6, step 460 and related text, "Create redirection stubs that will call trace entry and ext method around target method").

Claim 12:

Berkley also discloses a process according to claim 1, wherein: said particular purpose is to add a first tracer (see for example, Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…").

Claim 13:

Claim 13 is another version process and product for monitoring as in claim 1 addressed above, and further amended to add limitation "automatically determining which methods, of a set of methods call one or more other method and are synthetic". As addressed in claim 1 above, Webster discloses a method to selective trace (using or without using a "first tracing mechanism") method calls according to the users defined selection in their interests (see for example, col.2, lines 34-36, "A user can specify particular trace information of interest, and interpreter can then track for each called method whether or not trace information is to be provided"). It is also well-known in Java programming fields that the terms and concepts including Java method calls, and Java synthetic method are widely used in Java programming practice. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to selectively trace (using a first tracing mechanism or without using the first tracing mechanism) the specific method calls (method call other method and the method is synthetic method) for the particular diagnostic/debugging purposes and further considering system's performance by avoiding system generating

irrelevant information as suggested by Webster (see for example, col.1, lines 40-42, "The ability to track or monitor these method calls is very important to a Java programmer for diagnostic and debugging purpose" and col.2, lines 34-39, "A user can specify particular trace information of interest…avoid the system having to output large quantities of irrelevant information, thereby greatly assisting in performance").

Claim 20:

Berkley further discloses a process according to claim 13, wherein: said step of using a first tracing mechanism includes modifying existing object code to add said first tracing mechanism (see for example, Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…").

Claim 39:

Berkley discloses an apparatus capable of monitoring, comprising:

- means for automatically determining whether a method is call another method (see for example, Fig.1 and related text, "METHOD B", "METHOD C" and also see col.2, lines 39-40, "means for determining whether the function (method) is active for a class of the executable using the modified configuration settings"); and

- means for tracing said method for a particular purpose only if said method calls another method (see for example, col.2, lines 41-43, "means for automatically dynamically creating a redirection stub to insert the function for the class if the function is active for that class", also see Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…"),

but does not explicitly disclose automatically determining sufficient scope during method calls or determining the method is a synthetic method. However, it is also well-known in Java programming fields that the terms and concepts including Java method calls, scope of the methods and Java synthetic method are widely used in Java programming practice. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine and selectively trace the specific method calls (method call other method, the sufficient scope of the method calls and the method is synthetic method) for the particular diagnostic/debugging purposes and further considering system's performance by avoiding system generating irrelevant information as suggested by Webster (see for example, col.1, lines 40-42, "The ability to track or monitor these method calls is very important to a Java programmer for diagnostic and debugging purpose" and col.2, lines 34-39, "A user can specify particular trace information of interest…avoid the system having to output large quantities of irrelevant information, thereby greatly assisting in performance").

Claim 40:

Berkley discloses an apparatus capable of monitoring, comprising:

- a storage device (see for example, Fig.3, items 102, 103 "Main Storage", "External Storage Media" and related text); and
- one or more processors in communication with said storage device (see for example, Fig.3, item 104, "CPU 1…CPU N" and related text), said one or more processors perform a process comprising:
  - accessing a method (see for example Fig.5, step 350 and related text, also see, col.2, lines 36-37, "Further, the system includes means for running the application executable using the modified runtime configuration settings");
  - tracing said method for a particular purpose only if said method calls one or more different methods and can be called by a sufficient scope of one or more other methods (see for example, col.2, lines 41-43, "means for dynamically creating a redirection stub to insert the function for the class if the function is active for that class", also see Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…").

But Berkley does not disclose:

- determining whether said method calls one or more different methods and can be called by a sufficient scope of one or more other methods.

However, it is well known in the Java programming that JVM (Java Virtual Machine) specification defines method by using a block starting with the tag "Method" that contains the information about calling other methods in java bytecode. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method calls another method and can be called by a sufficient scope of one or more other methods by checking the method information in that block while running by JIT in JVM or other bytecode scanning tools. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 41:

Berkley discloses an apparatus according to claim 40, but does not explicitly disclose said step of determining whether to modify said method includes determining whether said method is non-synthetic. However, it is well known in the Java programming that all synthetic methods generated by Java compiler are flagged in the class file and thus are easily identified (see for example, a well-known and widely used Java Virtual Machine Specification). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention

was made to determine whether said method is non-synthetic by checking the synthetic attribute field in bytecode while being compiled by JIT, Hotspot runtime or other bytecode scanning tools. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 42:

Berkley further discloses an apparatus according to claim 40, but does not explicitly disclose said step of determining whether to modify aid method includes determining whether said method has an access level of public or package. However, it is well known in the Java programming that JVM specification (see for example, a well-known and widely used Java Virtual Machine Specification) defines a set of access flags in method_info structure which has a flag name "ACC_PUBLIC" for access level of public or package. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method has an access level of public or package by using JIT, Hotspot runtime or other bytecode scanning tools to check this flag to determining whether said method has an access level of public or package. One would have been motivated to do so to allow a user to trace

specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 44:

Berkley discloses an apparatus according to claim 40 and further discloses said process further includes modifying existing object code for said method in order to add a first tracing mechanism (see for example, col.2, lines 45-46, "inserting a function into an application executable without recompiling the executable.")

Claims 47 and 51-52:

Claims 47 and 51-52 are another version process for monitoring as in claims 1-3, 5 and 8 addressed above, wherein all claimed limitation functions have been addressed and/or set forth above. Therefore, they also would have been obvious.

10. Claims 9, 11, 19, 21-23, 28-30, 32, 37, 38 and 45-46 are rejected under 35 U.S.C. 103(a) as being unpatentable over Berkley (Berkley et al., US 6,351,843) in view of Webster (US6,738,965) in further view of Berry (Berkley et al., US 6,662,359).

Claim 9:

Berkley and Webster disclose a process according to claim 1, but does not explicitly disclose said step of modifying includes adding a timer for said method. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, "timestamp" and related text"). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 11:

Berkley discloses a process according to claim 10, wherein:

- said start code starts a tracing process (see for example, Fig.6, step 460 and related text, "Create redirection stubs that will call trace entry and ext method around target method");
- said exit code stops said tracing process (see for example, Fig.6, step 460 and related text, "Create redirection stubs that will call trace entry and ext method around target method");

- said exit code is positioned to be executed subsequent to original object code (see for example, Fig.6, step 470 and related text, “Remaining class construction flows”);

But Berkley does not disclose said steps of adding exit code including jump instruction, exception table and said step of adding an entry in said exception table. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses:

- said step of adding exit code includes adding an instruction to jump to said exit code from said original object code (see for example. Fig.8 steps 812-816 and related text, also see col.9, line 55- col.10, line 8,”a jump around inserted code”);
- said step of adding exit code includes adding an entry in an exception table; and (see for example. Fig.8 step 802 and related text “Modify the exception table”);
- said step of adding an entry in said exceptions table includes adding a new entry into said exceptions table for said method, said new entry indicates a range of indices corresponding to said original object code, said new entry includes a reference to said exit code and said new entry indicates that said new entry pertains to all types of exceptions (see for example. Fig.8 steps 812-816 and related text, also see col.9, line 55- col.10, line 8,”a jump around inserted code”);

Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to add jump instruction and maintain exception table for inserting function into an application executable at runtime. One Would have been motivated to integrated Berry's steps into Berkley's process to ensure that code which moved due to either insertions or deletions is correctly relocated and related references are adjusted as pointed out by Berry (See for example, Col.9, lines 55-58, "to ensure that code which is moved due to either insertions or deletions is correctly relocated and related references are adjusted")

Claim 19:

Berkley discloses a process according to claim 13, but does not explicitly disclose said step of modifying includes adding a timer for said method. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, "timestamp" and related text"). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which

the function is to be implemented")

Claim 21:

Berkley discloses a process according to claim 20, but does not explicitly disclose said step of modifying includes adding a timer for said method. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, "timestamp" and related text"). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claims 22-23, 28-30 and 32:

Claims 22-23, 28-30 and 32 claim one or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, which is the product version of the process claims as discussed in claims 1-3 and 5-11 above respectively. Therefore, these claims are obvious

over Berkley, Webster and berry, because it is well known in the computer art to practice and/or produce such a program product for carrying out the acts/steps of such process by a typical computer processor.

Claims 33-35, 37 and 38:

Claims 33-35, 37 and 38 claim one or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, said processor readable code for programming one or more processors to perform a process as discussed in claims 13-15, 17 and 19 above respectively. Therefore, these claims are obvious over Berkley and Berry, because it is well known in the computer art to practice and/or produce such a program product for carrying out the acts/steps of such process by a typical computer processor.

Claim 45:

Berkley discloses an apparatus according to claim 44 above, but does not disclose said first tracing mechanism includes a timer. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, “timestamp” and related text”). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace

specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented").

Claim 46:

Berkley discloses an apparatus according to claim 44 above, but does not disclose said step of tracing includes timing said method. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, "timestamp" and related text"). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented").

Conclusion

11. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

12. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Zheng Wei whose telephone number is (571) 270-1059 and Fax number is (571) 270-2059. The examiner can normally be reached on Monday-Thursday 8:00-15:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature of relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571- 272-1000.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Z. W./
Examiner, Art Unit 2192

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192